ARL-TN-0912 ● SEP 2018

US Army Research Laboratory

Multi-Source Information Amalgamation Prototype: A Fuzzy Logic Approach

by Drew Kogon, John Richardson, and Timothy Hanratty

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-0912 ● SEP 2018

US Army Research Laboratory

Multi-Source Information Amalgamation Prototype: A Fuzzy Logic Approach

by John Richardson and Timothy Hanratty
Computational and Information Sciences Directorate, ARL

Drew Kogon
University of Southern California, Los Angeles, CA

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
September 2018	Technical Note	1 July 2018–17 August 2018

4. TITLE AND SUBTITLE
Multi-Source Information Amalgamation Prototype: A Fuzzy Logic Approach

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Drew Kogon, John Richardson, and Timothy Hanratty

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CII-T
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0912

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
As part of its ongoing research into the calculation of the value of information, the US Army Research Laboratory has expanded its single source Fuzzy Associative Memory model. The original model only accounted for a single source of data and calculated value based on the parameters of the data. The new model accounts for multi-source data and considers the relationship (supporting or conflicting) between the data when calculating value. This report outlines a software implementation for calculating value in the multi-source model.

15. SUBJECT TERMS
Information amalgamation, fuzzy logic, Fuzzy Associative Memory, value of information, multi-source intelligence

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Timothy Hanratty
Unclassified	Unclassified	Unclassified	UU	38	19b. TELEPHONE NUMBER (Include area code) (410) 278-3084

Contents

List of Figures iv

1. Introduction 1

2. Background: Single-Source VoI Model 1

3. Multi-Source Architecture 3

3.1 Step 1: Applicability Score Calculation 4

3.2 Step 2: Cognitive Group Membership Scoring 6

3.3 Step 3: Merging the Information Scores 8

3.4 Step 4: Final Calculations and Contextual VoI 12

4. Conclusion and Future Directions 14

5. References 15

Appendix. Multi-Source Code 17

List of Symbols, Abbreviations, and Acronyms 31

Distribution List 32

List of Figures

Fig. 1 Single-source VoI model 1
Fig. 2 Fuzzy rule base example (tactical) 2
Fig. 3 Multi-source architecture overview 3
Fig. 4 Multi-source prototype example 4
Fig. 5 Applicability Score calculations 5
Fig. 6 COG FAM 7

1. Introduction

Today, military operations are defined by myriad information sources that provide an unprecedented volume, velocity, variety, and veracity of information not attained in most other domains. Given this wealth of information, a primary challenge for military commanders and their staff is separating the important information from the routine (US Army Headquarters 2003; Gates 2010). Calculating information importance, termed the value of information (VoI) metric, is a daunting task that is highly context dependent, requiring humans to judge the information's value within a host of differing operational situations (Alberts et al. 2001). The purpose of this report is to technically note the programming task associated with the "Multi-Source Information Amalgamation Prototype" (Hanratty et al. 2017a) completed during the summer of 2018.

2. Background: Single-Source VoI Model

A Fuzzy Associative Memory (FAM) model was chosen to construct the single-source VoI system. A FAM is a k-dimensional table where each dimension corresponds to one of the input universes of the rules. The *i*-th dimension of the table is indexed by the fuzzy sets that comprise the decomposition of the *i*-th input domain. Fuzzy if-then rules are represented within the FAM. While numerous characteristics could be applicable to determining VoI, the features of source reliability, information content, timeliness, and mission context were used as the starting point to construct the single-source VoI system. More detailed descriptions of the FAMs, the fuzzy rule bases, the domain decompositions, and other implementation aspects of the prototype system can be found in Hammell et al. (2012), Hanratty et al. (2013), Hanratty (2017), and Hanratty et al. (2017a, 2017b). The architecture of the single-source VoI system is shown in Fig. 1.

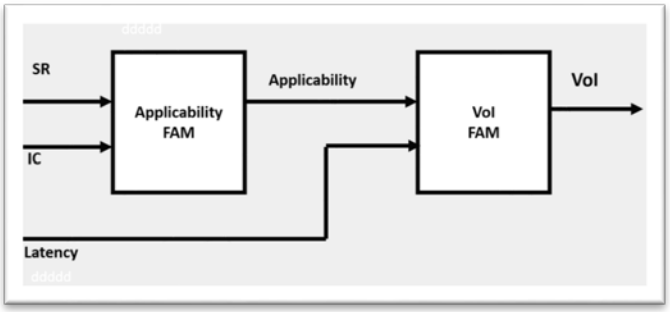

Fig. 1 Single-source VoI model

Two inputs feed into the Applicability FAM: source reliability (SR) and information content (IC); the output of this FAM is termed the information applicability metric. Likewise, two inputs feed into the VoI FAM: one of these (information applicability) is the output of the first FAM; the other input is the information latency rating. The output of the second FAM, and the overall system output, is the VoI metric.

The tactical version of the fuzzy rule base associated with the single-source VoI system is depicted in Fig. 2. The row and column indices of each FAM define a potential rule antecedent within the appropriate input domain. The number in each cell represents the consequent value of the rule that is represented by the cell indices.

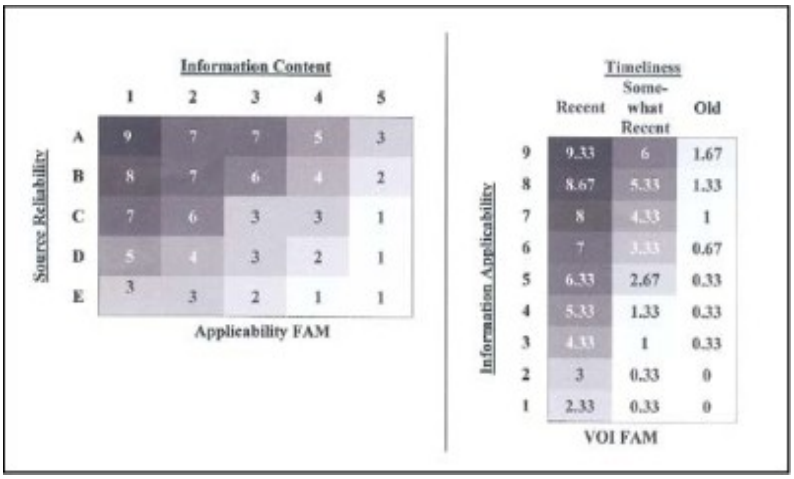

Fig. 2 Fuzzy rule base example (tactical)

The output from the system is determined by the standard centroid defuzzification strategy. That is, the degree to which each rule influences the overall output is directly related to the degree to which its inputs match its antecedent fuzzy sets. The degree of the *i*-th fuzzy rule, $\deg_{C^i}^{i}$, corresponding to inputs (x_1, y_1) is

$$\deg_{C^i}^{i} = m_{I_1^i}(x_1) m_{I_2^i}(y_1), \tag{1}$$

where C^i is the output region of Rule *i* and $m_{I_j^i}$ is the degree of membership of the input in the input region of Rule *i* for the *j*-th component.

The standard centroid defuzzification equation that is used to produce the overall output from a set of inputs (x_1, y_1) is

$$y = \frac{\sum_{i=1}^{k} \deg_{C^i}^{i}, mid^{i}}{\sum_{i=1}^{k} \deg_{C^i}^{i}} . \tag{2}$$

3. Multi-Source Architecture

Figure 3 shows an overview of the multi-source VoI architecture. Depicted in the shaded regions are the required extensions to the single-source in order to capture multi-source information.

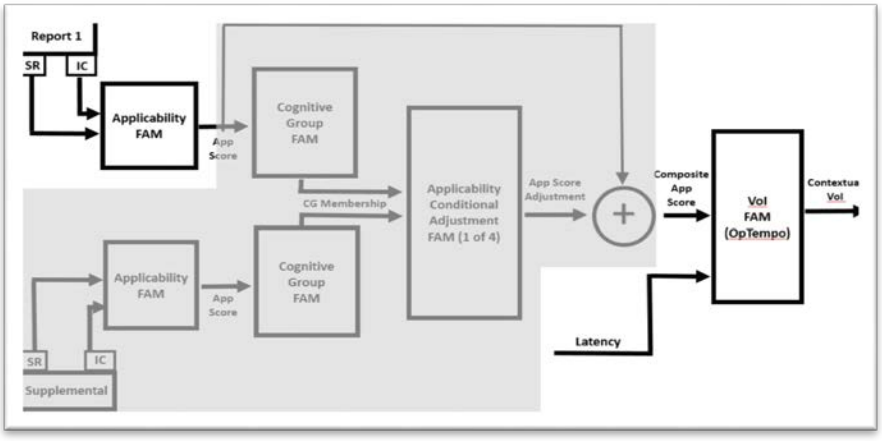

Fig. 3 Multi-source architecture overview

The new prototype adds two additional FAM operations:

- The Cognitive Group FAM (COG FAM) transforms the calculated Applicability Score into one of the five cognitive groups.
- The Applicability Conditional Adjustment FAM, given two cognitive groups, calculates the adjustment (higher for complementary or lower for contradictory) to the Applicability Score of the original piece of information by an amount commensurate to how well it complements or contradicts the original premise.

The following subsections walk through the code execution for each step of the multifaceted system. As an example, consider the situation illustrated by Fig. 4. In this case, the analyst receives two pieces of information. The initial information has an SR rating between D and C (denoted by C–), an IC rating valued at 3.5 (halfway

between 3 and 4), and a latency valued as “recent”. The supplemental information, on the other hand, is judged to totally support the initial information and is graded with an SR of B+, an IC of 1.5, and a latency of “somewhat recent”.

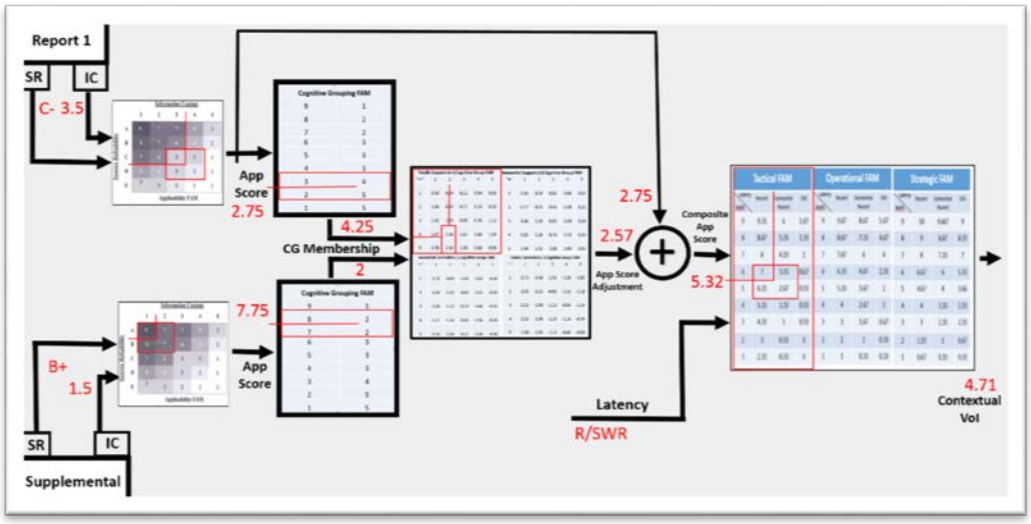

Fig. 4 Multi-source prototype example

3.1 Step 1: Applicability Score Calculation

Given the SR and IC for the initial and supplemental information, the first step involves calculating their respective **Applicability Scores**. Utilizing the standard centroid defuzzification strategies discussed in the previous section, the row and column indices of each FAM define a rule’s antecedent within the input domain and the number in each cell represents the consequent value of that rule. In this case, as shown in Fig. 5, the source reliability of the initial information partially matches the antecedents of two fuzzy sets, “C” and “D”, while the information content input partially matches the fuzzy sets for “3” and “4”. Similarly, the supplemental information matches the fuzzy sets associated with “A” and “B” and “1” and “2”. Following Eq. 2, the output of each fuzzy rule is scaled by the degree (membership) to which the antecedents match and result in scores of **2.75** for the initial information and **7.75** for the supplemental information.

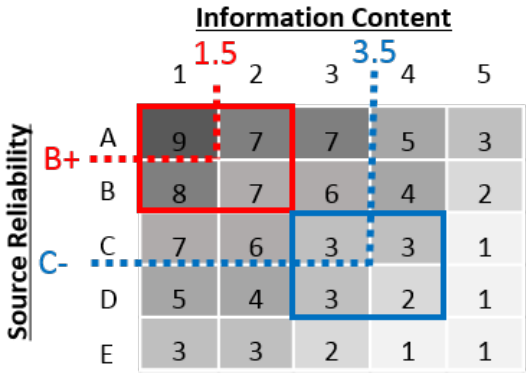

Fig. 5 Applicability Score calculations

Listed below is the function for the Applicability Score calculation where the inputs are SR and IC. Each value (supplemental and initial) is run through separately. This allows flexibility in the number of supplemental scores.

```
1.  public static double computeAPP(double SR, double IC) { //This is the functi
    on of App Score calculation where: //SR represents Source Reliability //IC r
    epresents Information Content
2.      double VOI, APP, IC1_deg, IC2_deg, SR1_deg, SR2_deg;
3.      double IC1, IC2, SR1, SR2;
4.      double[][] APPFAM = {
5.          {
6.              9.0, 7.0, 7.0, 5.0, 3.0
7.          }, {
8.              8.0, 7.0, 6.0, 4.0, 2.0
9.          }, {
10.             7.0, 6.0, 3.0, 3.0, 1.0
11.         }, {
12.             5.0, 4.0, 3.0, 2.0, 1.0
13.         }, {
14.             3.0, 2.0, 1.0, 1.0, 1.0
15.         }
16.     };
17.     int SRint = (int) Math.floor(SR);
18.     int ICint = (int) Math.floor(IC);
19.     if (SRint == 5) {
20.         SR1 = 4;
21.         SR2 = 5;
```

```
22.         SR1_deg = 0.0;
23.         SR2_deg = 1.0;
24.     } else {
25.         SR1 = SRint;
26.         SR2 = SR1 + 1;
27.         SR1_deg = (-1.0) * (SR - 1.0) + SRint;
28.         SR2_deg = 1 - SR1_deg;
29.     }
30.     if (ICint == 5) {
31.         IC1 = 4;
32.         IC2 = 5;
33.         IC1_deg = 0.0;
34.         IC2_deg = 1.0;
35.     } else {
36.         IC1 = ICint;
37.         IC2 = IC1 + 1;
38.         IC1_deg = (-1.0) * (IC - 1.0) + ICint;
39.         IC2_deg = 1 - IC1_deg;
40.     } //This represents the calculation of the degree that each  //value fal
    ls into a cell within the  //Information Content x Source Reliability Matrix
      //(Applicability FAM)
41.     APP = ((SR1_deg * IC1_deg * APPFAM[(int)(SR1 - 1)][(int)(IC1 - 1)])) + (
    SR1_deg * IC2_deg * APPFAM[(int)(SR1 - 1)][(int)(IC2 - 1)]) + (SR2_deg * IC1
    _deg * APPFAM[(int)(SR2 - 1)][(int)(IC1 - 1)]) + (SR2_deg * IC2_deg * APPFAM
    [(int)(SR2 - 1)][(int)(IC2 - 1)]);
42.     return APP;
43. };
```

3.2 Step 2: Cognitive Group Membership Scoring

For the second step, the calculated Applicability Scores are applied to the COG FAM to determine their membership among the cognitive groups. As can be seen in Fig. 6, the initial information (2.75 Applicability Score) causes two rules to fire, giving it a cognitive group membership of 4.25, and the supplemental information (7.75 Applicability Score) causes two rules to fire, giving it a cognitive group membership of 2.

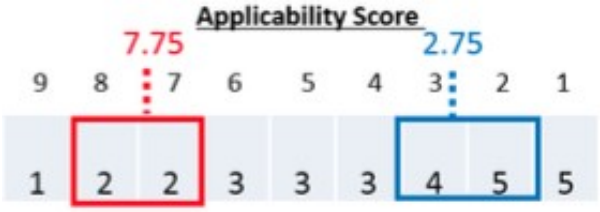

Fig. 6 COG FAM

For this section, we execute the computation by treating the COG FAM as a 1-D array and use the Applicability Scores to identify the index within the array. Following this, we perform the appropriate membership degree to obtain an average score. Outlined below is the function of "Cog Score" where APP represents the Applicability Score calculated previously. Each value (supplemental and initial) is run through the function separately. This allows flexibility in the number of supplemental scores.

```
1.  public static double computeCOG(double APP) { ///this section calculates the
     Cognitive Group FAM of an APP score input to the one dimensional COG FAM //
    Figure 8 of the paper
2.      double cog_group_membership, diff;
3.      double[] APP_SCORE = {
4.          5, 5, 4, 3, 3, 3, 2, 2, 1
5.      };
6.      double index_initial_infoAPP = APP - 1; //This section gets the top end
    of the index and the bottom end e.g App=3.2, bottom is 3, top is 4.
7.      int top_idx = (int) Math.ceil(index_initial_infoAPP);
8.      int bott_idx = (int) Math.floor(index_initial_infoAPP); //if the value i
    s the same in both indexes, then there's no point in finding a degree
9.      if (APP_SCORE[(int) top_idx] == APP_SCORE[(int) bott_idx]) {
10.         cog_group_membership = APP_SCORE[(int) top_idx];
11.     } else { //but if it isn't the same, need to find new value
12.         diff = APP_SCORE[(int) bott_idx] - APP_SCORE[(int) top_idx]; //this
    line below takes the floor score and adds the product of the difference scor
    e (almost certainly 1) and the inverse of the values to the right of the dec
    imal point //in other words if the app score is 2.75, it takes 5-
    4=1 and multiplies by the difference of 3-
    2.75 or .25, resulting in 4.25 as the correct score)
13.         cog_group_membership = APP_SCORE[(int) top_idx] + (diff * ((float) t
    op_idx - (float) index_initial_infoAPP));
14.     }; //System.out.print("Cog group membership:"+ cog_group_membership);
15.     return cog_group_membership;
16. };
```

3.3 Step 3: Merging the Information Scores

Now that we have obtained a Cognitive Group Membership for the Initial Statement (4.25) and the Supplemental Statement (2), the scores are used to obtain an Applicability Adjustment (APP ADJ) Score using one of four 5 × 5 Applicability Conditional Adjustment FAM matrices: 1) Totally Supports, 2) Somewhat Supports, 3) Somewhat Contradicts, and 4) Totally Contradicts.

Currently the code features a determiner variable that selects the appropriate APP ADJ FAM matrix when indicated, but has been hard-coded to select the "Totally Supports" APP ADJ FAM matrix.

Following is the function for the APP ADJ Score calculation where the output, "final score", represents the APP ADJ Score. The function takes three inputs:

- COG FAM for initial statement
- COG FAM for supplemental statement
- A determiner for APP ADJ FAM matrix selection

Similar to the previous step, we use the input scores as indices to locate the relevant FAM values. CORE VALUES matrix represents a maximum of four cells from the APP ADJ matrix that the two input scores cover. The function then collapses the scores horizontally, reducing the dimensions to a single array before collapsing vertically to obtain the final value. Since the APP ADJ matrices are not one-directional or evenly distributed, our code identifies the direction of the core values to ensure proper addition or subtraction of degree values.

```
1.  ////////////now adding in the APP ADJ FAM/////////////////////////// ///nee
    d branching statements for the level of support to determine matrix values
2.  public static double computeADJAPP(double cog_init, double cog_sup, int dete
    rminer) { ///the determiner is an input for the data of the relevant FAM mat
    rix
3.      if (determiner == 1) { //this is totally supports version
4.          double[][] APP_ADJFAM = {
5.              {.03, .04, .11, .04, .03
6.              }, {
7.                  1.60, .47, .71, .15, .35
8.              }, {
9.                  1.82, 1.59, .95, .70, 1.12
10.             }, {
11.                 2.67, 2.44, 1.61, 1.48, 1.05
```

```
            }, {
                3.78, 2.94, 1.93, .66, .58
            }
        };
    } else if (determiner == 2) { ///PLACEHOLDER //values needed for somewha
t supports
        double[][] APP_ADJFAM = {
            {.03, .04, .11, .04, .03
            }, {
                1.60, .47, .71, .15, .35
            }, {
                1.82, 1.59, .95, .70, 1.12
            }, {
                2.67, 2.44, 1.61, 1.48, 1.05
            }, {
                3.78, 2.94, 1.93, .66, .58
            }
        };
    } else if (determiner == 3) { ///PLACEHOLDER //values needed for somewha
t contradicts
        double[][] APP_ADJFAM = {
            {.03, .04, .11, .04, .03
            }, {
                1.60, .47, .71, .15, .35
            }, {
                1.82, 1.59, .95, .70, 1.12
            }, {
                2.67, 2.44, 1.61, 1.48, 1.05
            }, {
                3.78, 2.94, 1.93, .66, .58
            }
        };
    } else if (determiner == 4) { ///PLACEHOLDER //values needed for totally
 contradicts
        double[][] APP_ADJFAM = {
            {.03, .04, .11, .04, .03
```

```
            }, {
                1.60, .47, .71, .15, .35
            }, {
                1.82, 1.59, .95, .70, 1.12
            }, {
                2.67, 2.44, 1.61, 1.48, 1.05
            }, {
                3.78, 2.94, 1.93, .66, .58
            }
        };
    } else {
        double[][] APP_ADJFAM = { ///PLACEHOLDER ///this is a placeholder st
atement
            {.03, .04, .11, .04, .03
            }, {
                1.60, .47, .71, .15, .35
            }, {
                1.82, 1.59, .95, .70, 1.12
            }, {
                2.67, 2.44, 1.61, 1.48, 1.05
            }, {
                3.78, 2.94, 1.93, .66, .58
            }
        };
    }; /////Now that I have the matrix selected, it's time to take the two i
nputted COG values
    double bott_idx_column, top_idx_column, top_idx_row, bott_idx_row, row_i
dx, col_idx, compos_1 = 0, compos_2 = 0; ///supplement value
    cog_sup = cog_sup - 1; //initial cog value
    cog_init = cog_init - 1; ///one value represents the row value and the o
ther the column in this new matrix
    top_idx_row = Math.ceil(cog_init);
    bott_idx_row = Math.floor(cog_init);
    top_idx_column = Math.ceil(cog_sup);
    bott_idx_column = Math.floor(cog_sup); ////this is the placeholder FAM w
e have on file.
    double[][] APP_ADJFAM = {
```

```
77.         {.03, .04, .11, .04, .03
78.         }, {
79.             1.60, .47, .71, .15, .35
80.         }, {
81.             1.82, 1.59, .95, .70, 1.12
82.         }, {
83.             2.67, 2.44, 1.61, 1.48, 1.05
84.         }, {
85.             3.78, 2.94, 1.93, .66, .58
86.         }
87.     }; ///the core values are the maximum of 4 cells of the matrix that the
    two scores cover.
88.     double[][] CORE_VALUES = {
89.         {
90.             APP_ADJFAM[(int) bott_idx_row][(int) bott_idx_column], APP_ADJFA
    M[(int) bott_idx_row][(int) top_idx_column]
91.         }, {
92.             APP_ADJFAM[(int) top_idx_row][(int) bott_idx_column], APP_ADJFAM
    [(int) top_idx_row][(int) top_idx_column]
93.         }
94.     }; ///now its time to simplify the 2x2 to make a 1x2 //these two lines g
    ather the absolute value difference score between two cells in the matrix
95.     double row1_diff = Math.abs(CORE_VALUES[0][0] - CORE_VALUES[0][1]);
96.     double row2_diff = Math.abs(CORE_VALUES[1][0] - CORE_VALUES[1][1]); //th
    is is done because the direction is not linear for the tactical boxes so we
    need to determine which direction to take (e.g. add or subtract) //the diffe
    rence values from the main cell ///here we are taking the ceiling value of t
    he row and subtracting it from the actual to get the inverse of it e.g. 3-
    2.576= .434 which goes into the equation below for proper degree
97.     if (CORE_VALUES[0][0] > CORE_VALUES[0][1]) {
98.         compos_1 = CORE_VALUES[0][0] - row1_diff * ((float) top_idx_column -
     (float) cog_sup);
99.     } else if (CORE_VALUES[0][0] < CORE_VALUES[0][1]) {
100.         compos_1 = CORE_VALUES[0][0] + row1_diff * ((float) top_idx_column
    - (float) cog_sup);
101.     } else if (CORE_VALUES[0][0] == CORE_VALUES[0][1]) { //this line is for
     the lower dimension versions so if we dont get overlap on the first dimensi
    on
102.         compos_1 = CORE_VALUES[0][0];
103.     } ////now we do the same process for the second row to get one value
```

```
104.     if (CORE_VALUES[1][0] > CORE_VALUES[1][1]) {
105.         compos_2 = CORE_VALUES[1][0] - row2_diff * ((float) top_idx_column
    - (float) cog_sup);
106.     } else if (CORE_VALUES[1][0] < CORE_VALUES[1][1]) {
107.         compos_2 = CORE_VALUES[1][0] + row2_diff * ((float) top_idx_column
    - (float) cog_sup);
108.     } else if (CORE_VALUES[1][0] == CORE_VALUES[1][1]) {
109.         compos_2 = CORE_VALUES[1][0];
110.     } ///now onto the vertical integration stage where we take the two comp
    osites, compos_1 and compos_2 and combine them for the score
111.     double final_Score = 0, fin_Diff = 0, ratio_multiplier = 0; //this is t
    he final difference between the two scores
112.     fin_Diff = Math.abs(compos_1 - compos_2); //we need to know the ratio b
    y which the cognitive score enters the second cell e.g. 3.75, the overlap is
     .75
113.     ratio_multiplier = 1 - (top_idx_row - cog_init); //here we need to know
     the direction by which we include the difference ratio. It can either be su
    btracted or added
114.     if (compos_1 == compos_2) {
115.         final_Score = compos_1;
116.     } else if (compos_1 > compos_2) {
117.         final_Score = compos_1 - ratio_multiplier * fin_Diff;
118.     } else if (compos_1 < compos_2) {
119.         final_Score = compos_1 + ratio_multiplier * fin_Diff;
120.     }
121.     return final_Score;
122. };
```

3.4 Step 4: Final Calculations and Contextual VoI

After obtaining the APP ADJ score (2.57), we retrieve the original APP score (2.75) and create a composite APP score (2.57 + 2.75 = 5.32). Following is the function for the final VoI score, where "VOI" represents the output score. The function takes three inputs:

- APPcomposite: the composite APP ADJ score
- TIMinit: the initial latency score
- TIMsupp: the supplemental latency score

The two latency values are averaged to select the appropriate vertical index with the APP ADJ composite score serving as the horizontal index. Our code currently only executes from the Tactical VoI FAM since those were the only data available at the time of this construction. However, future development should include a tempo variable for selecting the other two VoI FAMs (Operational FAM and Strategic FAM), which are now available. In our example, latency receives a score of "Recent" and "Somewhat Recent". Compiled, this results in a combined Latency Score of **1.5**. When combined with the APP ADJ Score of **5.32**, our final result is a VoI Score of **4.32**.

```
1.  public static double computeVOI(double APPcomposite, double TIM_init, double
    TIM_supp) {
2.      double VOI, APP1_deg, APP2_deg, TIM1_deg, TIM2_deg, TIM;
3.      double APP1, APP2, TIM1, TIM2;
4.      TIM = Math.abs((TIM_init + TIM_supp) / 2);
5.      int APPint = (int) Math.floor(APPcomposite);
6.      int TIMint = (int) Math.floor(TIM);
7.      double[][] VOIFAM = {
8.          {
9.              2.33, 0.33, 0.0
10.         }, {
11.             3.0, 0.33, 0.0
12.         }, {
13.             4.33, 1.0, 0.33
14.         }, {
15.             5.33, 1.33, 0.33
16.         }, {
17.             6.33, 2.67, 0.33
18.         }, {
19.             7.0, 3.33, 0.67
20.         }, {
21.             8.0, 4.33, 1.0
22.         }, {
23.             8.67, 5.33, 1.33
24.         }, {
25.             9.33, 6.0, 1.67
26.         }
```

```
    };
    if (APPint == 9) {
        APP1 = 8;
        APP2 = 9;
        APP1_deg = 0.0;
        APP2_deg = 1.0;
    } else {
        APP1 = APPint;
        APP2 = APP1 + 1;
        APP1_deg = (-1.0) * (APPcomposite - 1.0) + APPint;
        APP2_deg = 1 - APP1_deg;
    }
    if (TIMint == 3) {
        TIM1 = 2;
        TIM2 = 3;
        TIM1_deg = 0.0;
        TIM2_deg = 1.0;
    } else {
        TIM1 = TIMint;
        TIM2 = TIM1 + 1;
        TIM1_deg = (-1.0) * (TIM - 1.0) + TIMint;
        TIM2_deg = 1 - TIM1_deg;
    }
    VOI = (APP1_deg * TIM1_deg * VOIFAM[(int)(APP1 - 1)][(int)(TIM1 - 1)]) +
     (APP1_deg * TIM2_deg * VOIFAM[(int)(APP1 - 1)][(int)(TIM2 - 1)]) + (APP2_de
    g * TIM1_deg * VOIFAM[(int)(APP2 - 1)][(int)(TIM1 - 1)]) + (APP2_deg * TIM2_
    deg * VOIFAM[(int)(APP2 - 1)][(int)(TIM2 - 1)]);
    return VOI;
}
```

4. Conclusion and Future Directions

Future work should incorporate the Strategic VoI and Operational VoI FAMs into this program. Pending empirical data, values for the “Somewhat Relevant”, “Somewhat Contradicts”, and “Totally Contradicts” Applicability Conditional Adjustment FAM matrices should be included. Finally, we recommend a GUI overlay for our current work for ease in testing and program usage.

5. References

Alberts DS, Garstka JJ, Hayes RE, Signori DA. Understanding information age warfare. Washington (DC): Office of the Assistant Secretary of Defense (C3I/Command Control Research Program); 2001.

Gates RM. Quadrennial defense review report. Washington (DC): Department of Defense; 2010.

Hammell RJ, Hanratty T, Heilman E. Capturing the value of information in complex military environments: a fuzzy-based approach. Proceedings of the 2012 IEEE International Conference on Fuzzy Systems; 2012 June 10–15; Brisbane, Australia. New York (NY): IEEE; c2003. p. 1–7.

Hanratty TP, Hammell II RJ, Bodt BA, Heilman EG, Dumer JC. Enhancing battlefield situational awareness through fuzzy-based value of information. 46th Hawaii International Conference on System Sciences (HICSS); 2013 Jan 7–10; Maui, HI. New York (NY): IEEE; c2013. p. 1402–1411.

Hanratty T, Heilman E, Richardson J, Caylor J. A fuzzy-logic approach to information amalgamation: a framework for human-agent collaboration. Proceedings of the 2017 IEEE International Conference on Fuzzy Systems; 2017a July 9–12; Naples, Italy. New York (NY): IEEE; c2017. p. 1–6.

Hanratty T, Heilman E, Richardson J, Mittrick M, Caylor J. Determining the perceived value of information when combining supporting and conflicting data. In: Next-Generation Analyst V; vol 10207. Proceedings of SPIE Defense + Security; 2017b May 3; Anaheim, CA. Bellingham (WA): SPIE. doi: 10.1117/12.2264820.

Headquarters, Department of the Army. Mission command: command and control of Army forces. Washington (DC): Headquarters, Department of the Army; 2003 Aug. Field Manual No.: FM 6-0.

INTENTIONALLY LEFT BLANK.

Appendix. Multi-Source Code

This appendix appears in its original form, without editorial change.

```
1.  import java.util.Random;
2.  public class VoIFull {
3.      private static int[][] cards = {
4.          {
5.              1, 1, 1
6.          }, {
7.              1, 1, 2
8.          }, {
9.              1, 1, 3
10.         }, {
11.             1, 2, 1
12.         }, {
13.             1, 2, 2
14.         }, {
15.             1, 2, 3
16.         }, {
17.             1, 3, 1
18.         }, {
19.             1, 3, 2
20.         }, {
21.             1, 3, 3
22.         }, {
23.             1, 4, 1
24.         }, {
25.             1, 4, 2
26.         }, {
27.             1, 4, 3
28.         }, {
29.             1, 5, 1
30.         }, {
31.             1, 5, 2
32.         }, {
33.             1, 5, 3
34.         }, {
35.             2, 1, 1
```

```
        }, {
            2, 1, 2
        }, {
            2, 1, 3
        }, {
            2, 2, 1
        }, {
            2, 2, 2
        }, {
            2, 2, 3
        }, {
            2, 3, 1
        }, {
            2, 3, 2
        }, {
            2, 3, 3
        }, {
            2, 4, 1
        }, {
            2, 4, 2
        }, {
            2, 4, 3
        }, {
            2, 5, 1
        }, {
            2, 5, 2
        }, {
            2, 5, 3
        }, {
            3, 1, 1
        }, {
            3, 1, 2
        }, {
            3, 1, 3
        }, {
```

```
            3, 2, 1
        }, {
            3, 2, 2
        }, {
            3, 2, 3
        }, {
            3, 3, 1
        }, {
            3, 3, 2
        }, {
            3, 3, 3
        }, {
            3, 4, 1
        }, {
            3, 4, 2
        }, {
            3, 4, 3
        }, {
            3, 5, 1
        }, {
            3, 5, 2
        }, {
            3, 5, 3
        }, {
            4, 1, 1
        }, {
            4, 1, 2
        }, {
            4, 1, 3
         }, {
             4, 2, 1
         }, {
             4, 2, 2
         }, {
             4, 2, 3
```

```
106.         }, {
107.             4, 3, 1
108.         }, {
109.             4, 3, 2
110.         }, {
111.             4, 3, 3
112.         }, {
113.             4, 4, 1
114.         }, {
115.             4, 4, 2
116.         }, {
117.             4, 4, 3
118.         }, {
119.             4, 5, 1
120.         }, {
121.             4, 5, 2
122.         }, {
123.             4, 5, 3
124.         }, {
125.             5, 1, 1
126.         }, {
127.             5, 1, 2
128.         }, {
129.             5, 1, 3
130.         }, {
131.             5, 2, 1
132.         }, {
133.             5, 2, 2
134.         }, {
135.             5, 2, 3
136.         }, {
137.             5, 3, 1
138.         }, {
139.             5, 3, 2
140.         }, {
```

```
141.             5, 3, 3
142.         }, {
143.             5, 4, 1
144.         }, {
145.             5, 4, 2
146.         }, {
147.             5, 4, 3
148.         }, {
149.             5, 5, 1
150.         }, {
151.             5, 5, 2
152.         }, {
153.             5, 5, 3
154.         }
155.     };
156.     public static void main(String[] args) {
157.         String newLine = System.getProperty("line.separator");
158.         Random rand = new Random();
159.         for (int i = 0; i < cards.length; i++) {
160.             double APP, VOI, APPsupp, cog_group_membership, cog_group_membe
    rship_supp, composite_ADJ_Score;
161.             int z = rand.nextInt(cards.length - 1) + 0;
162.             double SR = cards[i][0];
163.             double IC = cards[i][1];
164.             double TIM = cards[i][2];
165.             double SRsupp = cards[z][0];
166.             double ICsupp = cards[z][1];
167.             double TIMsupp = cards[z][2];
168.             APP = computeAPP(SR, IC); //random int generator for the supple
    ment
169.             APPsupp = computeAPP(SRsupp, ICsupp);
170.             cog_group_membership = computeCOG(APP);
171.             cog_group_membership_supp = computeCOG(APPsupp);
172.             composite_ADJ_Score = computeADJAPP(cog_group_membership, cog_g
    roup_membership_supp, 1);
173.             VOI = computeVOI(APP + composite_ADJ_Score, TIM, TIMsupp);
```

```
174.             System.out.print("I= " + i + newLine + "APP: " + APP + ", [" +
    SR + "," + IC + "," + TIM + "]" + newLine + "Supplemental APP: " + APPsupp +
     ", [" + SRsupp + "," + ICsupp + "," + TIMsupp + "]" + newLine);
175.             System.out.print("Cog Score " + cog_group_membership + newLine
    + "Cog Supplement Score: " + cog_group_membership_supp + newLine + "Adjusted
     Score: " + composite_ADJ_Score + newLine);
176.             System.out.print("VOI Final: " + VOI + newLine);
177.             System.out.print(newLine);
178.         }
179.     }
180.     public static double computeAPP(double SR, double IC) { //This is the f
    unction of App Score calculation where: //SR represents Source Reliability /
    /IC represents Information Content
181.         double VOI, APP, IC1_deg, IC2_deg, SR1_deg, SR2_deg;
182.         double IC1, IC2, SR1, SR2;
183.         double[][] APPFAM = {
184.             {
185.                 9.0, 7.0, 7.0, 5.0, 3.0
186.             }, {
187.                 8.0, 7.0, 6.0, 4.0, 2.0
188.             }, {
189.                 7.0, 6.0, 3.0, 3.0, 1.0
190.             }, {
191.                 5.0, 4.0, 3.0, 2.0, 1.0
192.             }, {
193.                 3.0, 2.0, 1.0, 1.0, 1.0
194.             }
195.         };
196.         int SRint = (int) Math.floor(SR);
197.         int ICint = (int) Math.floor(IC);
198.         if (SRint == 5) {
199.             SR1 = 4;
200.             SR2 = 5;
201.             SR1_deg = 0.0;
202.             SR2_deg = 1.0;
203.         } else {
204.             SR1 = SRint;
```

```
205.             SR2 = SR1 + 1;
206.             SR1_deg = (-1.0) * (SR - 1.0) + SRint;
207.             SR2_deg = 1 - SR1_deg;
208.         }
209.         if (ICint == 5) {
210.             IC1 = 4;
211.             IC2 = 5;
212.             IC1_deg = 0.0;
213.             IC2_deg = 1.0;
214.         } else {
215.             IC1 = ICint;
216.             IC2 = IC1 + 1;
217.             IC1_deg = (-1.0) * (IC - 1.0) + ICint;
218.             IC2_deg = 1 - IC1_deg;
219.         } //This represents the calculation of the degree that each  //valu
    e falls into a cell within the  //Information Content x Source Reliability M
    atrix  //(Applicability FAM)
220.         APP = ((SR1_deg * IC1_deg * APPFAM[(int)(SR1 - 1)][(int)(IC1 - 1)])
    ) + (SR1_deg * IC2_deg * APPFAM[(int)(SR1 - 1)][(int)(IC2 - 1)]) + (SR2_deg
    * IC1_deg * APPFAM[(int)(SR2 - 1)][(int)(IC1 - 1)]) + (SR2_deg * IC2_deg * A
    PPFAM[(int)(SR2 - 1)][(int)(IC2 - 1)]);
221.         return APP;
222.     }
223.     public static double computeCOG(double APP) { ///this section calculate
    s the Cognitive Group FAM of an APP score input to the one dimensional COG F
    AM //Figure 8 of the paper
224.             double cog_group_membership, diff;
225.             double[] APP_SCORE = {
226.                 5, 5, 4, 3, 3, 3, 2, 2, 1
227.             };
228.             double index_initial_infoAPP = APP - 1; //This section gets the
     top end of the index and the bottom end e.g App=3.2, bottom is 3, top is 4.
229.             int top_idx = (int) Math.ceil(index_initial_infoAPP);
230.             int bott_idx = (int) Math.floor(index_initial_infoAPP); //if th
    e value is the same in both indexes, then there's no point in finding a degr
    ee
231.             if (APP_SCORE[(int) top_idx] == APP_SCORE[(int) bott_idx]) {
232.                 cog_group_membership = APP_SCORE[(int) top_idx];
233.             } else { //but if it isn't the same, need to find new value
```

```
234.                 diff = APP_SCORE[(int) bott_idx] - APP_SCORE[(int) top_idx]
    ; //this line below takes the floor score and adds the product of the differ
    ence score (almost certainly 1) and the inverse of the values to the right o
    f the decimal point //in other words if the app score is 2.75, it takes 5-
    4=1 and multiplies by the difference of 3-
    2.75 or .25, resulting in 4.25 as the correct score)
235.                 cog_group_membership = APP_SCORE[(int) top_idx] + (diff * (
    (float) top_idx - (float) index_initial_infoAPP));
236.             }; //System.out.print("Cog group membership:"+ cog_group_member
    ship);
237.             return cog_group_membership;
238.         } ////////////now adding in the APP ADJ FAM//////////////////////
    //// ///need branching statements for the level of support to determine matr
    ix values
239.     public static double computeADJAPP(double cog_init, double cog_sup, int
     determiner) { ///the determiner is an input for the data of the relevant FA
    M matrix
240.         if (determiner == 1) { //this is totally supports version
241.             double[][] APP_ADJFAM = {
242.                 {.03, .04, .11, .04, .03
243.                 }, {
244.                     1.60, .47, .71, .15, .35
245.                 }, {
246.                     1.82, 1.59, .95, .70, 1.12
247.                 }, {
248.                     2.67, 2.44, 1.61, 1.48, 1.05
249.                 }, {
250.                     3.78, 2.94, 1.93, .66, .58
251.                 }
252.             };
253.         } else if (determiner == 2) { ///PLACEHOLDER //values needed for so
    mewhat supports
254.             double[][] APP_ADJFAM = {
255.                 {.03, .04, .11, .04, .03
256.                 }, {
257.                     1.60, .47, .71, .15, .35
258.                 }, {
259.                     1.82, 1.59, .95, .70, 1.12
260.                 }, {
261.                     2.67, 2.44, 1.61, 1.48, 1.05
```

```
                }, {
                    3.78, 2.94, 1.93, .66, .58
                }
            };
        } else if (determiner == 3) { ///PLACEHOLDER //values needed for somewhat contradicts
            double[][] APP_ADJFAM = {
                {.03, .04, .11, .04, .03
                }, {
                    1.60, .47, .71, .15, .35
                }, {
                    1.82, 1.59, .95, .70, 1.12
                }, {
                    2.67, 2.44, 1.61, 1.48, 1.05
                }, {
                    3.78, 2.94, 1.93, .66, .58
                }
            };
        } else if (determiner == 4) { ///PLACEHOLDER //values needed for totally contradicts
            double[][] APP_ADJFAM = {
                {.03, .04, .11, .04, .03
                }, {
                    1.60, .47, .71, .15, .35
                }, {
                    1.82, 1.59, .95, .70, 1.12
                }, {
                    2.67, 2.44, 1.61, 1.48, 1.05
                }, {
                    3.78, 2.94, 1.93, .66, .58
                }
            };
        } else {
            double[][] APP_ADJFAM = { ///PLACEHOLDER ///this is a placeholder statement
                {.03, .04, .11, .04, .03
```

```
                }, {
                    1.60, .47, .71, .15, .35
                }, {
                    1.82, 1.59, .95, .70, 1.12
                }, {
                    2.67, 2.44, 1.61, 1.48, 1.05
                }, {
                    3.78, 2.94, 1.93, .66, .58
                }
            };
        }; /////Now that I have the matrix selected, it's time to take the two inputted COG values
        double bott_idx_column, top_idx_column, top_idx_row, bott_idx_row, row_idx, col_idx, compos_1 = 0, compos_2 = 0; ///supplement value
        cog_sup = cog_sup - 1; //initial cog value
        cog_init = cog_init - 1; ///one value represents the row value and the other the column in this new matrix
        top_idx_row = Math.ceil(cog_init);
        bott_idx_row = Math.floor(cog_init);
        top_idx_column = Math.ceil(cog_sup);
        bott_idx_column = Math.floor(cog_sup); ////this is the placeholder FAM we have on file.
        double[][] APP_ADJFAM = {
            {.03, .04, .11, .04, .03
            }, {
                1.60, .47, .71, .15, .35
            }, {
                1.82, 1.59, .95, .70, 1.12
            }, {
                2.67, 2.44, 1.61, 1.48, 1.05
            }, {
                3.78, 2.94, 1.93, .66, .58
            }
        }; ///the core values are the maximum of 4 cells of the matrix that the two scores cover.
        double[][] CORE_VALUES = {
            {
```

```
                APP_ADJFAM[(int) bott_idx_row][(int) bott_idx_column], APP_
ADJFAM[(int) bott_idx_row][(int) top_idx_column]
            }, {
                APP_ADJFAM[(int) top_idx_row][(int) bott_idx_column], APP_A
DJFAM[(int) top_idx_row][(int) top_idx_column]
            }
        }; ///now its time to simplify the 2x2 to make a 1x2 //these two li
nes gather the absolute value difference score between two cells in the matr
ix
        double row1_diff = Math.abs(CORE_VALUES[0][0] - CORE_VALUES[0][1]);

        double row2_diff = Math.abs(CORE_VALUES[1][0] - CORE_VALUES[1][1]);
 //this is done because the direction is not linear for the tactical boxes s
o we need to determine which direction to take (e.g. add or subtract) //the
difference values from the main cell ///here we are taking the ceiling value
 of the row and subtracting it from the actual to get the inverse of it e.g.
 3-2.576= .434 which goes into the equation below for proper degree
        if (CORE_VALUES[0][0] > CORE_VALUES[0][1]) {
            compos_1 = CORE_VALUES[0][0] - row1_diff * ((float) top_idx_col
umn - (float) cog_sup);
        } else if (CORE_VALUES[0][0] < CORE_VALUES[0][1]) {
            compos_1 = CORE_VALUES[0][0] + row1_diff * ((float) top_idx_col
umn - (float) cog_sup);
        } else if (CORE_VALUES[0][0] == CORE_VALUES[0][1]) { //this line is
 for the lower dimension versions so if we dont get overlap on the first dim
ension
            compos_1 = CORE_VALUES[0][0];
        } ////now we do the same process for the second row to get one valu
e
        if (CORE_VALUES[1][0] > CORE_VALUES[1][1]) {
            compos_2 = CORE_VALUES[1][0] - row2_diff * ((float) top_idx_col
umn - (float) cog_sup);
        } else if (CORE_VALUES[1][0] < CORE_VALUES[1][1]) {
            compos_2 = CORE_VALUES[1][0] + row2_diff * ((float) top_idx_col
umn - (float) cog_sup);
        } else if (CORE_VALUES[1][0] == CORE_VALUES[1][1]) {
            compos_2 = CORE_VALUES[1][0];
        } ///now onto the vertical integration stage where we take the two
composites, compos_1 and compos_2 and combine them for the score
        double final_Score = 0, fin_Diff = 0, ratio_multiplier = 0; //this
is the final difference between the two scores
        fin_Diff = Math.abs(compos_1 - compos_2); //we need to know the rat
io by which the cognitive score enters the second cell e.g. 3.75, the overla
p is .75
```

```
350.         ratio_multiplier = 1 - (top_idx_row - cog_init); //here we need to
    know the direction by which we include the difference ratio. It can either b
    e subtracted or added
351.         if (compos_1 == compos_2) {
352.             final_Score = compos_1;
353.         } else if (compos_1 > compos_2) {
354.             final_Score = compos_1 - ratio_multiplier * fin_Diff;
355.         } else if (compos_1 < compos_2) {
356.             final_Score = compos_1 + ratio_multiplier * fin_Diff;
357.         }
358.         return final_Score;
359.     };
360.     public static double computeVOI(double APPcomposite, double TIM_init, d
    ouble TIM_supp) {
361.         double VOI, APP1_deg, APP2_deg, TIM1_deg, TIM2_deg, TIM;
362.         double APP1, APP2, TIM1, TIM2;
363.         TIM = Math.abs((TIM_init + TIM_supp) / 2);
364.         int APPint = (int) Math.floor(APPcomposite);
365.         int TIMint = (int) Math.floor(TIM);
366.         double[][] VOIFAM = {
367.             {
368.                 2.33, 0.33, 0.0
369.             }, {
370.                 3.0, 0.33, 0.0
371.             }, {
372.                 4.33, 1.0, 0.33
373.             }, {
374.                 5.33, 1.33, 0.33
375.             }, {
376.                 6.33, 2.67, 0.33
377.             }, {
378.                 7.0, 3.33, 0.67
379.             }, {
380.                 8.0, 4.33, 1.0
381.             }, {
382.                 8.67, 5.33, 1.33
```

```
            }, {
                9.33, 6.0, 1.67
            }
        };
        if (APPint == 9) {
            APP1 = 8;
            APP2 = 9;
            APP1_deg = 0.0;
            APP2_deg = 1.0;
        } else {
            APP1 = APPint;
            APP2 = APP1 + 1;
            APP1_deg = (-1.0) * (APPcomposite - 1.0) + APPint;
            APP2_deg = 1 - APP1_deg;
        }
        if (TIMint == 3) {
            TIM1 = 2;
            TIM2 = 3;
            TIM1_deg = 0.0;
            TIM2_deg = 1.0;
        } else {
            TIM1 = TIMint;
            TIM2 = TIM1 + 1;
            TIM1_deg = (-1.0) * (TIM - 1.0) + TIMint;
            TIM2_deg = 1 - TIM1_deg;
        }
        VOI = (APP1_deg * TIM1_deg * VOIFAM[(int)(APP1 - 1)][(int)(TIM1 - 1
    )]) + (APP1_deg * TIM2_deg * VOIFAM[(int)(APP1 - 1)][(int)(TIM2 - 1)]) + (AP
    P2_deg * TIM1_deg * VOIFAM[(int)(APP2 - 1)][(int)(TIM1 - 1)]) + (APP2_deg *
    TIM2_deg * VOIFAM[(int)(APP2 - 1)][(int)(TIM2 - 1)]);
        return VOI;
    }
}
```

List of Symbols, Abbreviations, and Acronyms

1-D	1-dimensional
APP ADJ	Applicability Adjustment
COG FAM	Cognitive Group FAM
FAM	Fuzzy Associative Memory
GUI	graphical user interface
IC	information content
SR	source reliability
VoI	value of information

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIR ARL IMAL HRA RECORDS MGMT RDRL DCL TECH LIB
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
3 (PDF)	ARL RDRL CII T T HANRATTY E HEILMAN R HOBBS